E	7	6	5	4	3	2	1	0	Description
1	0	0	0	1	1	0	1	1	USER_DATA JPEG
0	v	v	v	v	v	v	v	v	This Token informs the Video Demux that the DATA Token that follows contains user data. See A.11.3, "Conversion of start codes to Tokens", and A.14.6, "Receiving User and Extension data". During JPEG operation the 8 bit field carries the JPEG marker value. This allows the class of user data to be identified.
0	0	0	0	1	1	0	1	1	USER_DATA MPEG
									This Token informs the Video Demux that the DATA Token that follows contains user data. See A.11.3, "Conversion of start codes to Tokens", and A.14.6, "Receiving User and Extension data"
1	1	1	1	0	1	1	0	1	VBV_BUFFER_SIZE
1	r	r	r	r	r	r	s	s	s - a 10 bit integer as defined by MPEG.
0	s	s	s	s	s	s	s	s	
1	1	1	1	0	1	1	1	0	VBV_DELAY
1	b	b	b	b	b	b	b	b	b - a 16 bit integer as defined by MPEG
0	b	b	b	b	b	b	b	b	
1	1	1	1	1	1	1	0	1	VERTICAL_MBS
1	r	r	r	v	v	v	v	v	v - a 13 bit integer indicating the vertical size of the picture in macroblocks
1	r	r	r	v	v	v	v	v	
1	1	1	1	1	0	0	1	1	VERTICAL_SIZE
1	v	v	v	v	v	v	v	v	v - a 16 bit integer indicating the vertical size of the picture in pixels.
0	v	v	v	v	v	v	v	v	This can be any integer value.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 9 of 9)

A.4.6 Signal levels

The two-wire interface uses CMOS inputs and output. V_{1Hmin} is approx. 70% of V_{DD} and V_{1Lmax} is approx. 30% of V_{DD}. The values shown in Table A.4.3 are those for V_{1H} and V_{1L} at their respective worst case V_{DD}. V_{DD}=5.0∀0.25V.

Symbol	Parameter	Min.	Max.	Units
V_{IH}	Input logic '1' voltage	3.68	V_{DD} - 0.5	V
V_{IL}	Input logic 'O' voltage	GND - 0.5	1.43	V
V_{OH}	Output logic '1' voltage	V_{DD} - 0.1		V^a
		V_{DD} - 0.4		V^b
V_{OL}	Output logic '0' voltage		0.1	V^c
			0.4	V^d
I_{IN}	Input leakage current		± 10	ΦA

Table A.4.3 DC electrical characteristics

a. I_{OH}#1mA

b. I_{OH}#4mA

c. I_{OL}#1mA

d. I_{OL}#4mA

A.5.5 Interface registers

Register name	Size/Dir.	Reset State	Description
interface_timing_access	1 bit rw	0	This function enable register allows access to the DRAM interface timing configuration registers. The configuration registers should not be modified while this register requests access to modify the configuration registers. After a 0 has been written to this register the DRAM interface will start to use the new values in the timing configuration registers.
page_start_length	5 bit rw	0	Specifies the length of the access start in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 32 ticks.
transfer_cycle_length	4 bit rw	0	Specifies the length of the fast page read or write cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
refresh_cycle_length	4 bit rw	0	Specifies the length of the refresh cycle in ticks. The minimum value that can be used is 4. (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
RAS_falling	4 bit rw	0	Specifies the number of ticks after the start of the access start that $\overline{RAS}$ falls. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
CAS_falling	4 bit rw	8	Specifies the number of ticks after the start of a read cycle, write cycle or access start that $\overline{CAS}$ falls. The minimum value that can be used is 1 (meaning 1 tick). 0 selects the maximum length of 16 ticks.

Table A.5.2 Interface timing configuration registers

(See Table A.9.7).

When configured for Token input via the MPI, the current Token is extended with the current value of coded_extn each time a value is written into coded_data[7:0]. Software is responsible for setting coded_extn to 0 before the last word of any Token is written to coded_data[7:0].

For example, a DATA Token is started by writing 1 into coded_extn and then 0x04 into coded_data[7:0]. The start of this new DATA Token then passes into the Spatial Decoder for processing.

Each time a new 8 bit value is written to coded_data[7:0], the current Token is extended. Coded_extn need only be accessed again when terminating the current Token, e.g. to introduce another Token. The last word of the current Token is indicated by writing 0 to coded_extn followed by writing the last word of the current Token into coded_data[7:0].

Register name	Size\Dir.	Reset State	Description
coded_extn	1 rw	x	Tokens can be supplied to the Spatial Decoder via the MPI by writing to these registers.
coded_data[7:0]	8 w	x	
coded_busy	1 r	1	The state of this registers indicates if the Spatial Decoder is able to accept Tokens written into coded_data [7:0]. The value 1 indicates that the interface is busy and unable to accept data. Behaviour is undefined if the user tries to write to coded_data [7:0] when coded_busy = 1.
enable_mpi-input	1 rw	0	The value in this function enable registers controls whether coded data input to the Spatial Decoder is via the coded data port (0) or via the MPI (1).

Table A.10.2 Coded data input registers

Each time before writing to coded_data[7:0], coded_busy should be inspected to see if the interface is ready to accept more data.

A.10.3 Switching between input modes

Provided suitable precautions are observed, it is possible to dynamically change the data input mode. In general, the transfer of a Token via any one route should be completed before switching modes.

Previous Mode	Next Mode	Behaviour
Byte	Token	The on-chip circuitry will use the last byte supplied in byte mode as the last byte of the DATA Token that it was constructing (i.e. the extn bit will be set to 0). Before accepting the next token.
	MPI input	

Table A.10.3 Switching data input modes

Previous Mode	Next Mode	Behaviour
Token	Byte	The off-chip circuitry supplying the Token in Token mode is responsible for completing the Token (i.e. with the extn bit of the last byte of information set to 0) before selecting byte mode.
	MPI input	Access to input via the MPI will not be granted (i.e. coded_busy will remain set to 1) until the off-chip circuitry supplying the Token in Token mode has completed the Token (i.e. with the extn bit of the last byte of information set to 0).
MPI input	Byte	The control software must have completed the Token (i.e. with the extn bit of the last byte of information set to 0) before enable_mpi_input is set to 0.
	MPI input	

Table A.10.3 Switching data input modes (contd)

The first byte supplied in byte mode causes a DATA Token header to be generated on-chip. Any further bytes transferred in byte mode are thereafter appended to this DATA Token until the input mode changes. Recall, DATA Tokens can contain as many bits as are necessary.

The MPI register bit, coded busy, and the signal, coded_accept, indicate on which interface the Spatial decoder is willing to accept data. Correct observation of these signals ensures that no data is lost.

A.10.4 Rate of accepting coded data

In the present invention, the input circuit passes Tokens to the Start Code Detector (see section A.11). The Start code Detector analyses data in the DATA Tokens bit serially. The Detector's normal rate of

Register name	Size/Dir.	Reset State	Description
start_code_detector_access	1 rw	0	Writing 1 to this register requests that the start code detector stop to allow access to its registers. The user should wait until the value 1 can be read from this register indicating that operation has stopped and access is possible.

Table A.11.1 Start code detector Registers (Sheet 1 of 5)

Register Name	Size/Dir.	Reset State	Description
unrecognised_start_event	1 rw	0	If an unrecognised start code is encountered this event will occur. If the mask register is set to 1 then an interrupt can be generated and the start code detector will stop. The start code value read from the bitstream is available in the register start_value while the start code detector is halted. See A.11.4.3 During normal operation start_value contains the value of the most recently decoded start/marker code. Only the 4 LSBs of start_value are used during H.261 operation. The 4 MSBs will be zero.
unrecognised_start_mask	1 rw	0	
start_value	8 ro	x	
stop_after_picture_event	1 rw	0	If the register stop_after_picture is set to 1 then a stop after picture event will be generated after the end of a picture has passed through the start code detector. If the mask register is set to 1 then an interrupt can be generated and the start code detector will stop. See A.11.5.1 stop_after_picture does not reset to 0 after the end of a picture has been detected so should be cleared directly.
stop_after_picture_mask	1 rw	0	
stop_after_picture	1 rw	0	

Table A.11.1 Start code detector Registers (Sheet 3 of 5)

Register Name	Size/Dir.	Reset State	Description
non_aligned_start_event	1 rw	0	When ignore_non_aligned is set to 1, start codes that are not byte aligned are ignored (treated as normal data) When ignore_non_aligned is set to 0, H.261 and MPEG start codes will be detected regardless of byte alignment and the non-aligned start event will be generated. If the mask register is set to 1 then the event will cause an interrupt and the start code detector will stop. See A.11.6. If the coding standard is configured as JPEG Ignore_non_aligned is ignored and the non-aligned start event will never be generated.
non_aligned_start_mask	1 rw	0	
Ignore_non_aligned	1 rw	0	
discard_extension_data	1 rw	0	When these registers are set to 1 extension or user data that cannot be decoded by the Spatial Decoder is discarded by the start code detector. See A.11.3.3.
discard_user_data	1 rw	0	
discard_all_data	1 rw	0	When set to 1 all data and Tokens are discarded by the start code detector. This continues until a **FLUSH** Token is supplied or the register is set to 0 directly. The **FLUSH** Token that resets this register is discarded and not output by the start code detector. See A.11.5.1 .
Insert_sequence_start	1 rw		See A.11.7

Table A.11.1 Start code detector Registers (Sheet 4 of 5)

Register Name	Size/Dir.	Reset State	Description
start_code_search	3 rw	5	When this register is set to 0 the start code detector operates normally. When set to a higher value the start code detector discards data until the specified type of start code is detected. When the specified start code is detected the register is set to 0 and normal operation follows. See A.11.8 .
start_code_detector_coding_standard	2 rw	0	This register configures the coding standard used by the start code detector. The register can be loaded directly or by using a **CODING_STANDARD** Token. Whenever the start code detector generates a **CODING_STANDARD** Token (see A.11.7.4 on page 109) it carries its current coding standard configuration. This Token will then configure the coding standard used by all other parts of the decoder chip-set. See A.21.1 on page 180 and A.11.7.
picture_number	4 rw	0	Each time the start coded detector detects a picture start code in the data stream (or the H.261 or JPEG equivalent) a PICTURE_START Token is generated which carries the current value of picture_number. This register then increments.

Table A.11.1 Start code detector Registers (Sheet 5 of 5)

A.12.4 Start-up control registers

Register name	Size/Dir.	Reset State	Description
startup_access *CED_BS_ACCESS*	1 rw	0	Writing 1 to this register requests that the bit counter and gate opening logic stop to allow access to their configuration registers.
bit_count *CED_BS_COUNT*	8 rw	0	This bit counter is incremented as coded data leaves the start code detector. The number of bits required to increment bit_count once is approx. $2^{(bit\_count\_prescale+1)}$ x 512. The bit counter starts counting bits after a **FLUSH** Token passes through the bit counter. It is reset to zero and then stops incrementing after the bit count target has been met.
bit_count_prescale *CED_BS_PRESCALE*	3 rw	0	
bit_count_target *CED_BS_TARGET*	8 rw	x	This register specifies the bit count target. A target met event is generated whenever the following condition becomes true: bit_count>=bit_count_target
target_met_event *BS_TARGET_MET_EVENT*	1 rw	0	When the bit count target is met this event will be generated. If the mask register is set to 1 then an interrupt can be generated, however, the bit counter will NOT stop processing data. This event will occur when the bit counter increments to its target. It will also occur if a target value is written which is less than or equal to the current value of the bit counter. Writing 0 to bit_count_target will always generate a target met event
target_met_mask	1 rw	0	

Table A.12.1 Decoder start-up registers

Register Name	Size/Dir.	Reset State	Description
counter_flushed_event *BS_FLUSH_EVENT*	1 rw	0	When a FLUSH Token passes through the bit count circuit this event will occur. If the mask register is set to 1 then an interrupt can be generated and the bit counter will stop.
counter_flushed_mask	1 rw	0	
counter_flushed_too_early_event BS_FLUSH_BEFORE_TARGET_MET_EVENT	1 rw	x	If a **FLUSH** Token passes through the bit count circuit board and the bit count target has not been met this event will occur. If the mask register is set to 1 then an interrupt can be generated and the bit counter will stop. See A.12.10.
counter_flushed-too-early-mask	1 rw	0	
offchip_queue *CED_BS_QUEUE*	1 rw	0	Setting this register to 1 configures the gate opening logic to require microprocessor support. When t his register is set to 0 the output gate control logic will automatically control the operation of the output gate. See sections A.12.6 and A.12.7.
enable_stream *CED_BS_ENABLE_NXT_STM*	1 rw	0	When an off-chip queue is in use writing to enable_stream controls the behaviour of the output gate after the end of a stream passes through it. A one in this register enables the output gate to open. The register will be reset when an accept_enable interrupt is generated.

Table A.12.1 Decoder start-up registers (contd)

Register Name	Size/Dir.	Reset State	Description
accept_enable_event *BS_STREAM_END_EVENT*	1 rw	0.00	This event indicates that a **FLUSH** Token has passes through the output gate (causing it to close) and that an enable was available to allow the gate to open. If the mask register is set to 1 then an interrupt can be generated and the register enable_stream will be reset. See A.12.7.1 .
accept_enable_mask	1 rw	0.00	

Table A.12.1 Decoder start-up registers (contd)

consideration when polling such registers as cdb_full and cdb_empty to monitor buffer conditions.

Register name	Size/Dir.	Reset State	Description
buffer_manager_access	1 rw	1	This access bit stops the operation of the buffer manager so that its various registers can be accessed reliably. See A.6.4.1. Note: this access register is unusual as its default state after reset is 1. i.e. after reset the buffer manager is halted awaiting configuration via the microprocessor interface.
buffer_manager_keyhole_address	6 rw	x	Keyhole access to the extended address space used for the buffer manager registers shown below. See A.6.4.3 for more information about accessing registers through a keyhole.
buffer_manager_keyhole_data	8 rw	x	
buffer_limit	18 rw	x	This specifies the overall size of the DRAM array attached to the Spatial Decoder. All buffer addresses are calculated MOD this buffer size and s will wrap round within the DRAM provided.
cdb_base tb_base	18 rw	x	These registers point to the base of the coded data (cdb) and Token (tb) buffers.
cdb_length tb_length	18 rw	x	These registers specify the length (i.e. size) of the coded data (cdb) and Token (tb) buffers.
cdb_read tb_read	18 ro	x	These registers hold an offset from the buffer base and indicate where data will be read from next.
cdb_number tb_number	18 ro	x	These registers show how much data is currently held in the buffers.
cdb_full tb_full	1 ro	x	These registers will be set to 1 if the coded data (cdb) or Token (tb) buffer fills.
cdb_empty tb_empty	1 ro	x	These registers will be set to 1 if the coded data (cdb) or Token (tb) buffer empties.

Table A.13.1 Buffer manager registers

SECTION A.14 Video Demux

The Video Demux or Video parser as it is also called, completes the task of converting coded data into Tokens started by the Start Code Detector. There are four main processing blocks in the Video Demux: Parser State Machine, Huffman decoder (including an ITOD), Macroblock counter and ALU.

The Parser or state machine follows the syntax of the coded video data and instructs the other units. The Huffman decoder converts variable length coded (VLC) data into integers. The Macroblock counter keeps track of which section of a picture is being decoded. The ALU performs the necessary arithmetic calculations.

A.14.1 Video Demux registers

Register name	Size/Dir.	Reset State	Description
demux_access *CED_H_CTRL[7]*	1 rw	0	This access bit stops the operation of the Video Demux so that it's various registers can be accessed reliably. See A.6.4.1.
huffman_error_code *CED_H_CTRL[6:4]*	3 ro		When the Video Demux stops following the generation of a huffman_event interrupt request this 3 bit register holds a value indicating why the interrupt was generated. See A.14.5.1.
parser_error_code *CED_H_DMUX_ERR*	8 ro		When the Video Demux stops following the generation of a parser_event interrupt request this 8 bit register holds a value indicating why the interrupt was generated. See A.14.5.2 .
demux_keyhole_address *CED_H_KEYHOLE_ADDR*	12 rw	x	Keyhole access to the Video Demux's extended address space. See A.6.4.3 for more information about accessing registers through a keyhole.
demux_keyhole_data *CED_H_KEYHOLE*	8 rw	x	Tables A.14.2, A.14.3 and A.14.4 describe the registers that can be accessed via the keyhole.

Table A.14.1 Top level Video Demux registers

Register name	Size/Dir.	Reset State	Description
dummy_last_picture *CED_H_ALU_REG0* *r_rom_control* *r_dummy_last_frame_bit*	1 rw	0	When this register is set to 1 the Video Demux will generate information for a "dummy" intra picture as the last picture of an MPEG sequence. This function is useful when the Temporal Decoder is configured for automatic picture re-ordering (see A.18.3.5, "Picture sequence re-ordering") to flush the last P or I picture out of the Temporal Decoder. No "dummy" picture is required if: • the Temporal Decoder is not configured for re-ordering • another MPEG sequence will be decoded immediately (as this will also flush out the last picture) • the coding standard is not MPEG
field_info *CED_H_ALU_REG0* *r_rom_control* *r_field_info_bit*	1 rw	0	When this register is set to 1 the first byte of any MPEG extra_information_picture is placed in the FIELD_INFO Token. See A.14.7.1.
continue *CED_H_ALU_REG0* *r_rom_control* *r_field_continue_bit*	1 rw	0	This register allows user software to control how much extra, user or extension data it wants to receive when it is detected by the decoder. See A.14.6 and A.14.7 .
rom_revision *CED_H_ALU_REG1* *r_rom_revision*	8 ro		Immediately following reset this holds a copy of the microcode ROM revision number. This register is also used to present to control software data values read from the coded data. See A.14.6, "Receiving User and Extension data", on page 148 and A.14.7, "Receiving Extra Information".

Table A.14.1 Top level Video Demux registers (contd)

Register name	Size/Dir.	Reset State	Description
huffman_event	1 rw	0	A Huffman event is generated if an error is found in the coded data. See A.14.5.1 for a description of these events. If the mask register is set to 1 then an interrupt can be generated and the Video Demux will stop. If the mask register is set to 0 then no interrupt is generated and the Video Demux will attempt to recover from the error.
huffman_mask	1 rw	0	
parser_event	1 rw	0	A Parser event can be in response to errors in the coded data or to the arrival of information at the Video Demux that requires software intervention. See A.14.5.2 for a description of these events. If the mask register is set to 1 then an interrupt can be generated and the Video Demux will stop. If the mask register is set to 0 then no interrupt is generated and the Video Demux will attempt to continue.
parser_mask	12 rw	x	

Table A.14.1 Top level Video Demux registers (contd)

Register name	size/dir.	Reset State	Description
component_name_0 component_name_1 component_name_2 component_name_3	8 rw	x	During JPEG operation the register component_name_*n* holds an 8 bit value indicating (to an application) which colour component has the component ID *n*.
horiz_pels	16 rw	x	These registers hold the horizontal and vertical dimensions of the video being decoded in pixels. See section A.14.2 .
vert_pels	16 rw	x	
horiz_macroblocks	16 rw	x	These registers hold the horizontal and vertical dimensions of the video being decoded in macroblocks. See section A.14.2 .
vert_macroblocks	16 rw	x	

Table A.14.2 video demux picture construction registers

Register name	Size/Dir.	Reset State	Description
max_h	2 rw	x	These registers hold the macroblock width and height in blocks (8 x 8 pixels). The values 0 to 3 indicate a width/height of 1 to 4 blocks. See section A.14.2 .
max_v	2 rw	x	
max_component_id	2 rw	x	The values 0 to 3 indicate that 1 to 4 different video components are currently being decoded. See section A.14.2 .
Nf	8 rw	x	During JPEG operation this register holds the parameter Nf (number of image components in frame).
blocks_h_0 blocks_h_1 blocks_h_2 blocks_h_3	2 rw	x	For each of the 4 colour components the registers blocks_h_*n* and blocks_v_*n* hold the number of blocks horizontally and vertically in a macroblock for the colour component with component ID *n*. See section A.14.2 .
blocks_v_0 blocks_v_1 blocks_v_2 blocks_v_3	2 rw	x	
tq_0 tq_1 tq_2 tq_3	2 rw	x	The two bit value held by the register tq_*n* describes which inverse Quantisation table is to be used when decoding data with component ID *n*.

Table A.14.2 video demux picture construction registers (contd)

Register Name	Size/Dir.	Reset State	Description
dc_huff_0 dc_huff_1 dc_huff_2 dc_huff_3	2 rw		The two bit value held by the register dc_huff_*n* describes which Huffman decoding table is to be used when decoding the DC coefficients of data with component ID *n*. Similarly ac_huff_*n* describes the table to be used when decoding AC coefficients. Baseline JPEG requires up to two Huffman tables per scan. The only tables implemented are 0 and 1.
ac_huff_0 ac_huff_1 ac_huff_2 ac_huff_3	2 rw		
dc_bits_0[15:0] dc_bits_1[15:0]	8 rw		Each of these is a table of 16, eight bit values. They provide the BITS information (see JPEG Huffman table specification) which form part of the description of two DC and two AC Huffman tables. See section A.14.3.1 .
ac_bits_0[15:0] ac_bits_1[15:0]	8 rw		
dc_huffval_0[11:0] dc_huffval_1[1:0]	8 rw		Each of these is a table of 12, eight bit values. They provide the HUFFVAL information (see JPEG Huffman table specification) which form part of the description of two AC Huffman tables. See section A.14.3.1 .
ac_huffval_0(161:0) ac_huffval_1(161:0)	8 rw		Each of these is a table of 162, eight bit values. They provide the HUFFVAL information (see JPEG Huffman table specification) which form part of the description of two DC Huffman tables. See section A.14.3.1 .
dc_zssss_0 dc_zssss_1	8 rw		These 8 bit registers hold values that are "special cased" to accelerate the decoding of certain frequency used JPEG VLCs. dc_ssss - magnitude of DC coefficient is 0. ac_eob - end of block ac_zrl - run of 16 zeros
ac_eob_0 ac_eob_1	8 rw		
ac_zrl_0 ac_zrl_1	8 rw		

Table A.14.3 Video demux Huffman table registers

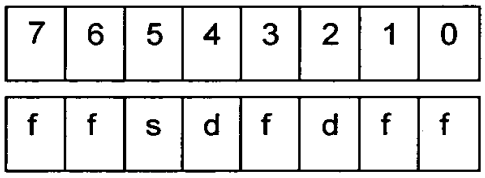
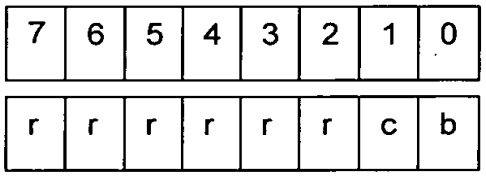
Register Name	Size/Dir.	Reset State	Description
picture_type	2 rw		During MPEG operation this register holds the picture type of the picture being decoded.
h_261_pic_type	8 rw		This register is loaded when decoding H.261 data. It holds information about the picture format.  Flags: s - Split Screen Indicator d - Document Camera r - Freeze Picture Release This value is not used by the decoder chips. However, the information should be used when configuring horiz_pels, vert_pels and the display or output device.
broken_closed	2 rw		During MPEG operation this register holds the broken_link and closed_gap information for the group of pictures being decoded.  Flags: C – closed_gap

Table A.14.4 Other Video Demux registers (contd)

A.15.2.7 Inverse quantizer test registers

Register Name	Size/Dir.	Reset State	Description
iq_quant_scale	5 rw		This register holds the current value of the quantisation scale factor. It is loaded by the QUANT_SCALE Token. This is not used during JPEG operation
iq_component	2 rw		This register holds the two bit component ID taken from the most recent DATA token head. This value is involved in the selection of the quantiser table. The register will also hold the table ID after a QUANT_TABLE Token arrives to load the table.
iq_prediction_mode	2 rw		This holds the two LSBs of the most recent PREDICTION_MODE Token.
iq_jpeg_indirection	8 rw		This register relates the two bit component ID number of a DATA Token to the table number of the quantisation table that should be used. Bits 1:0 specify the table number that will be sued with component 0 Bits 3:2 specify the table number that will be sued with component 1 Bits 5:4 specify the table number that will be sued with component 2 Bits 7:6 specify the table number that will be sued with component 3. This register is loaded by JPEG_TABLE_SELECT Tokens.
iq_mpeg_indirection	8 rw	0.00	This two bit register records whether to use default or down loaded quantisation tables with the intra and non-intra data. A 0 in the bit position indicates that the default table should be used.A.1 indicates that a down loaded table should be used. Bit 0 refers to intra data. Bit 1 refers to non-intra data. This register is normally loaded by the Token MPEG_TABLE_SELECT.

Table A.15.4 Inverse quantizer test registers

A.17.1 Temporal Decoder Signals

Signal Name	I/O	Pin Number	Description
in_data[8:0]	I	173, 172, 171, 169, 168, 167, 166, 164, 163	Input Port. This is a standard two wire interface normally connected to the Output Port of the Spatial Decoder. See sections A.4 and A.181
in_extn	I	174	
in_valid	I	162	
in_accept	O	161	
$\overline{enable}$ [1:0]	I	126, 127	Micro Processor Interface (MPI) See A.6.1. on page 69
$\overline{rw}$	I	125	
addr[7:0]	I	137, 136, 135, 133, 132, 131, 130, 128	
data[7:0]	O	152, 151, 149, 147, 145, 143, 141, 140	
irq	O	154	
DRAM_data[31:0]	I/O	15, 17, 19, 20, 22, 25, 27, 30, 31, 33, 35, 38, 39, 42, 44, 47, 49, 57, 59, 61, 63, 66 68, 70, 72, 74, 76, 79, 81, 83, 84, 85	DRAM Interface. See section A.5.2
DRAM_addr[10:0]	O	184, 186, 188, 189, 192, 193, 195, 197, 199, 200, 203	
$\overline{RAS}$	O	11	
$\overline{CAS}$ [3:0]	O	2, 4, 6, 8	
$\overline{WE}$	O	12	
$\overline{OE}$	O	204	
DRAM_enable	I	112	
out_data[7:0]	O	89, 90, 92, 93, 94, 95, 97, 98	Output Port. this is a standard two wire interface. See sections A.4
out_extn	O	87	
out_valid	O	99	
out_accept	I	100	
tck	I	115	JTAG port. See section A.8
tdi	I	116	
tdo	O	120	
tms	I	117	
$\overline{trst}$	I	121	
decoder_clock	I	177	The main decoder clock. See Table
$\overline{reset}$	I	160	Reset.

Table A.17.1 Temporal Decoder signals (contd)

Signal Name	I/O	Pin Num.	Description
tph0ish	I	122	If override = 1 then tph0ish and tph1ish are inputs for the on-chip two phase clock. For normal operation set override = 0. tph0ish and tph1ish are ignored (so connect to GND or VDD).
tph1ish	I	123	
override	I	110	
chiptest	I	111	Set chiptest = 0 for normal operation.
tloop	1	114	Connect to GND or VDD during normal operation.
ramtest	I	109	If ramtest = 1 test of the on-chip RAMs is enabled. Set ramtest = 0 for normal operation.
pllselect	I	178	If pllselect = 0 the on-chip phase locked loops are disabled.
ti	I	180	Two clocks required by the DRAM interface during test operation.
tq	I	179	
pdout	O	207	These two pins are connections for an external filter for the phase lock loop.
Pdin	1	206	

Table A.17.2 Temporal Decoder Test signals

Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	Pin
nc	208	nc	156	nc	104	nc	52
test pin	207	nc	155	nc	103	nc	51
test pin	206	irq	154	nc	102	nc	50
GND	205	nc	153	VDD	101	DRAM_data[15]	49
OE	204	data[7]	152	out_accept	100	nc	48
DRAM_addr[0]	203	data[6]	151	out_valid	99	DRAM_data[16]	47
VDD	202	nc	150	out_data[0]	98	nc	46
nc	201	data[5]	149	out_data[1]	97	GND	45
DRAM_addr[1]	200	nc	148	GND	96	DRAM_data[17]	44
DRAM_addr[2]	199	data[4]	147	out_data[2]	95	nc	43
GND	198	GND	146	out_data[3]	94	DRAM_data[18]	42
DRAM_addr[3]	197	data[3]	145	out_data[4]	93	VDD	41
nc	196	nc	144	out_data[5]	92	nc	40

Table A.17.3 Temporal Decoder Pin Assignments

Register Name	Size/Dir.	Reset State	Description
chip_access	1	1	Writing 1 to chip_access requests that the Temporal Decoder halt operation to allow re-configuration. The Temporal Decoder will continue operating normally until it reaches the end of the current video sequence. After reset is removed chip_access=1 i.e. the Temporal Decoder is halted.
chip_stopped_event	1	0	
chip_stopped_mask	1	0	When the chip stops a chip stopped event will occur. If chip_stopped_mask = 1 an interrupt will be generated.
count_error_event	1 rw	0	The Temporal Decoder has an adder that adds predictions to error data. If there is a difference between the number of error data bytes and the number of prediction data bytes then a count error event is generated. If count_error_mask = 1 an interrupt will be generated and prediction forming will stop. This event should only arise following a hardware error.
count_error_mask	1 rw		
picture_buffer_0	18	x	These specify the base addresses for the picture buffers.
picture_buffer_1	18	x	
component_offset_0	17	x	These specify the offset from the picture buffer pointer at which each of the colour components is stored. Data with component ID = n is stored starting at the position indicated by component_offset_n. See A.3.5.1, "Component Identification number".
component_offset_1	17 rw	x	
component_offset_2	17	x	
MPEG_recording	1 rw	0	Setting this register to 1 makes the Temporal Decoder change the picture order from the non-causal MPEG picture sequence to the correct display order by the. See A.18.3.5 . This register should is ignored during JPEG and H.261 operation.

Table A.18.2 Temporal Decoder registers

SECTION A.20 Late Write DRAM Interface

The interface is configurable in two ways:

The detail timing of the interface can be configured to accommodate a variety of different DRAM types

The "width" of the DRAM interface can be configured to provide a cost/performance trade-off

Signal Name	Input/ Output	Description
DRAM_data[31:0]	I/O	The 32 bit wide DRAM data bus. Optionally this bus can be configured to be 16 or 8 bits wide.
DRAM_addr[10:0]	O	The 22 bit wide DRAM interface address is time multiplexed over this 11 bit wide bus.
$\overline{RAS}$	O	The DRAM Row Address Strobe signal
$\overline{CAS}[3:0]$	O	The DRAM Column Address Strobe signal. One signal is provided per byte of the interface's data bus. All the $\overline{CAS}$ signals are driven simultaneously.
$\overline{WE}$	O	The DRAM Write Enable signal
$\overline{OE}$	O	The DRAM Output Enable signal
DRAM_enable	I	This input signal, when low, makes all the output signals on the interface go high impedance and stops activity on the DRAM interface

Table A.20.1 DRAM interface signals

Register name	size/ dir.	Reset State	Description
Modify_DRAM_timing	1 bit rw	0	This function enable register allows access to the DRAM interface timing configuration registers. The configuration registers should not be modified while this register holds the values zero. Writing a one to this register requests access to modify the configuration registers. After a zero has been written to this register the DRAM interface will start to use the new values in the timing configuration registers.

page_start_length	5 bit rw	0.00	Specifies the length of the access start in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 32 ticks.
read_cycle_length	4 bit rw	0.00	Specifies the length of the fast page read cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
write_cycle_length	4 bit rw	0.00	Specifies the length of the fast page late write cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
refresh_cycle_length	4 bit rw	0.00	Specifies the length of the refresh cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
RAS_falling	4 bit rw	0.00	Specifies the number of ticks after the start of the access start that falls. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
CAS_falling	4 bit rw	8	Specifies the number of ticks after the start of a read cycle, write cycle or access start that $\overline{CAS}$ falls. The minimum value that can be used is 1 (meaning 1 tick). 0 selects the maximum length of 16 ticks.
DRAM_data_width	2 bit rw	0.00	Specifies the number of bits used on the DRAM interface data bus DRAM_data[31:0]. See A.20.4 .
row_address_bits	2 bit rw	0.00	Specifies the number of bits used for the row address portion of the DRAM interface address bus. See A.20.5 .
DRAM_enable	1 bit rw	1	Writing the value 0 in to this register forces the DRAM interface into a high impedance state. 0 will be read from this register if either the DRAM_enable signal is low or 0 has been written to the register.

Table A.20.2 DRAM Interface configuration registers (contd)

internal queue consuming the queue. Similarly, stream_end_event is a request to supply the down stream queue; stream_end_event resets ced_bs_enable_nxt_stream. The two events should be serviced as follows:

```
/* TARGET_MET_EVENT */
j= micro_read(CED_BS_ENABLE_NEXT_STM);
if (j == 0) /*Is next stream enabled ?*/
{/*no,  enable it*/
micro_write(CED_BS_ENABLE_NXT_STM,  1);
printf(" enable next stream  (queue  =  0x%x))0 \n",  (context->queue));
}
else /*yes, increment the queue of "target_met" streams*/
{
queue++;
printf(" stream already enabled (queue  =  0x%x) \n", (context-
>queue));
}
/*  STREAM_EVENT  */
if (queue > 01) /*are there any "target_mets" left? */
{/*yes, decrement the que and enable another stream */
queue--;
 micro_write (CED_BS_ENABLE_NXT_STM, 1);
 printf(* enable next stream (queue  =  0x%x) \n*, (context->queue));
}
else
printf("  queue empty cannot enable next stream  (queue  =  0x%x) \n",
queue);
micro_write(CED_EVENT_1,  1  <<  BS_STREAM_END_EVENT);  /** clear
event
*/
```

Register Name	Address	Bits	Reset State	Function
BU_BM_ACCESS	0x10	[0]	1	Access bit for buffer manager
BU_BM_CTL0	0x11	[0] [1]	1 1	Max buf isb: 1->3 buffers 0->2 External picture clock select
BU_BM_TARGET_IX	0x12	[3:0]	0x0	For detecting arrival of picture
BU_BM_PRESS_NUM	0x13	[7:0]	0x00	Presentation number
BU_BM_THIS_PNUM	0x14	[7:0]	0xFF	Current picture number
BU_BM_PIC_NUM0	0x15	[7:0]	none	Picture number in buffer 1
BU_BM_PIC_NUM1	0x16	[7:0]	none	Picture number in buffer 2
BU_BM_PIC_NUM2	0x17	[7:0]	none	Picture number in buffer 3
BU_BM_TEMP_REF	0x18	[4:0]	0x00	Temporal reference from stream

Table C.2.3 User-Accessible Registers

Register Name	Address	Bits	Reset State	Function
BU_BM_PRES_FLAG	0x80	[0]	0.00	Presentation flag
BU_BM_EXP_TR	0x81	[4:0]	0xFF	Expected temporal reference
BU_BM_TR_DELTA	0x82	[4:0]	0x00	Delta
BU_BM_ARR_IX	0x83	[1:0]	0x0	Arrival buffer index
BU_BM_DSP_IX	0x84	[1:0]	0x0	Display buffer index
BU_BM_RDY_IX	0x85	[1:0]	0x0	Ready buffer index
BU_BM_BSTATE3	0x86	[1:0]	0x0	Buffer 3 status
BU_BM_BSTATE2	0x87	[1:0]	0x0	Buffer 2 status
BU_BM_BSTATE1	0x88	[1:0]	0x0	Buffer 1 status
BU_BM_INDEX	0x89	[1:0]	0x0	Current buffer index
BU_BM_STATE	0x8A	[4:0]	0x00	Buffer manager state
BU_BM_FROMPS	0x8B	[0]	0x0	From PICTURE_START flag
BU_BM_FROMFL	0x8C	[0]	0x0	From FLUSH_TOKEN flag

Table C.2.4 Test Registers

quantities are in the range of (32..470). Since the input to the **Top-Level Registers** CSC is Y, C_R, C_B, only the third and fourth of these equations are of relevance.

In the CSC design, the precision of the coefficients was chosen so that, for 9 bit data, all output values were within plus or minus 1 bit of the values produced by a full floating point simulation of the algorithm (this is the best accuracy that it is possible to achieve). This gave 13 bit twos-complement coefficients for cxO-cx3 and 14 bit twos-complement coefficients for cx4. The coefficients for all the design conversions are given below in both decimal and hex.

Table C.8.1 Coefficients for Various Conversions

	E_R - > Y		R - > Y		Y - > E_R		Y - > R	
Coeff	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
c01	0.299	0132	0.256		1.0	0400	1.169	04AD
c02	0.587	0259	0.502		1.402	059C	1.639	068E
c03	0.114	0075	0.098		0.00	0.00	0.00	0.00
c04	0.00	0.00	16		-179.46	F4C8	-228.48	F1B8
c11	0.5	0200	0.428		1.0	0400	1.169	04AD
c12	-0.42	FE53	-0.36		-0.71	FD25	-0.84	FCA9
c13	-0.08	FFAD	-0.07		-0.34	FEA0	-0.40	FE64
c14	128.0	0800	128		135.5	0878	139.7	08BA
c21	-0.17	FF53	-0.14		1.0	0400	1.169	04AD
c22	-0.33	FEAD	-0.28		0.00	0.00	0.00	0.00
c23	0.5	0200	0.427		1.772	0717	2.071	0849
c24	128	0800	128		-226.82	F1D2	-283.84	EE42

All these numbers are calculated from the fundamental equation:

$$Y = O.299E_R+O.587E_G+O.O114E_B$$

and the following color-difference equations:

$$C_R=E_R-Y$$

$$C_B=E_B-Y$$

REGISTER NAME	ADDRESS	BITS	COMMENTS
BU_BM_BSTATE1	0x88	2	R/W
BU_BM_INDEX	0x89	2	R/W
BU_BM_STATE	0x8a	5	R/W
BU_BM_FROMPS	0x8b	1	R/W
BU_BM_FROMFL	0x8c	1	R/W
BU_DA_COMP0_SNP3	0x90	8	R/W - These are the three snoopers on the display address generators address output
BU_DA_COMP0_SNP2	0x91	8	
BU_DA_COMP0_SNP1	0x92	8	
BU_DA_COMP0_SNP0	0x93	8	
BU_DA_COMP1_SNP3	0x94	8	
BU_DA_COMP1_SNP2	0x95	8	
BU_DA_COMP1_SNP1	0x96	8	
BU_DA_COMP1_SNP0	0x97	8	
BU_DA_COMP2_SNP3	0x98	8	
BU_DA_COMP2_SNP2	0x99	8	
BU_DA_COMP2_SNP1	0x9a	8	
BU_DA_COMP2_SNP0	0X9b	8	
BU_UV_RAM1A_ADDR_1	0xa0	8	R/W - upi test access into the vertical upsamplers' RAMs
BU_UV_RAM1A_ADDR_0	0xa1	8	
BU_UV_RAM1A_DATA	0xa2	8	
BU_UV_RAM1B_ADDR_1	0xa4	8	
BU_UV_RAM1B_ADDR_0	0xa5	8	
BU_UV_RAM1B_DATA	0xa6	8	
BU_UV_RAM2A_ADDR_1	0xa8	8	
BU_UV_RAM2A_ADDR_0	0xa9	8	
BU_UV_RAM2A_DATA	0xaa	8	
BU_UV_RAM2B_ADDR_1	0xac	8	
BU_UV_RAM2B_ADDR_0	0xad	8	
BU_UV_RAM2B_DATA	0xae	8	
BU_WA_ADDR_SNP1	0xb0	8	R/W - snooper on the write
BU_WA_ADDR_SNP0	0xb1	8	address generator address
BU_WA_ADDR_SNP0	0xb2	8	o/p
BU_WA_DATA_SNP1	0xb4	8	R/W - snooper on data
BU_WA_DATA_SNP0	0xb5	8	output of WA

Table C.11.1 Top-Level Registers A Top Level Address Map (contd)

Keyhole Register Name	Keyhole Address	Bits	Comments
BU_IF_SNP0_1	0xb8	8	R/W - Three snoopers on the dramif data outputs
BU_IF_SNP0_0	0xb9	8	
BU_IF_SNP1_1	0xba	8	
BU_IF_SNP1_0	0xbb	8	
BU_IF_SNP2_1	0xbc	8	
BU_IF_SNP2_0	0xbd	8	
BU_IFRAM_ADDR_1	0xc0	1	R/W - upi access it IF RAM
BU_IFRAM_ADDR_0	0xc1	8	
BU_IFRAM_DATA	0xc2	8	
BU_OC_SNP_3	0xc4	8	R/W – snooper on output of chip
BU_OC_SNP_2	0xc5	8	
BU_OC_SNP_1	0xc6	8	
BU_OC_SNP_0	0xc7	8	
BU_YAPLL_CONFIG	0xc8	8	R/W
BU_BM_FRONT_BYPASS	0xca	1	R/W

Table C.11.1 Top-Level Registers A

Top Level Address Map (contd)

C.12.1 Address Generator Keyhole Space

Notes on address generator keyhole table:

1)All registers in the address generator keyhole take up 4 bytes of address space regardless of their width. The missing addresses (0x00, 0x04 etc.) will always read back zero.

2)The access bit of the relevant block (dispaddr or waddrgen) must be set before accessing this keyhole.

```
if (hmbs_event)
  load(mbs_wide);
else if (vmbs_event)
  load(mbs_high);
else if (def_samp0_event)
{
  load (maxhb[0]);
  load (maxvb[0]);
}
else if (def_samp1_event)
{
  load (maxhb[1]);
  load (maxvb[1]);
}
else if (def_samp2_event)
{
  load (maxhb[2]);
  load (maxvb[2]);
}
```

In addition, the following calculations are necessary to retain consistent picture size parameters:

```
if (hmbs_event | | vmbs_event | |
    def_samp0_event | | def_samp1_event | | def_samp2_event)
{
 for (i=0; i<max_component; i++)
 {
  hbs[i] = addr_hbs[i] = (maxhb[i] + 1)  * mbs_wide;
  half_width_in_blocks[i] = ((maxhb[i] +1)  * mbs_wide)/2;
  last_mb_in_row[i]  =  hbs[i] - (maxhb[i] +1);
  last_mb_in_half_row[i]  =  half_width_in_blocks[i]  -
(maxhb[i] +1);
  last_row_in_mb[i]  =  hbs[i]  × maxvb[i];
  blocks_per_mb_row[i]  =  last_row_in_mb[i]  +  hbs[i];
  last_mb_row[i]  =  blocks_per_mb_row[i]  *  (mbs_high-1);
 }
```